Structural and Functional Dependence of Objects in Data Bases

Dipl.-Ing. J. Perevalova, Technical University of Berlin, 13355 Berlin, Germany
(julia@mail.bv.tu-berlin.de)
Prof.Dr.Dr.h.c.mult P. J. Pahl, Technical University of Berlin, 13355 Berlin, Germany
(pahl@ifb.bv.tu-berlin.de)

Summary
Let the information of a civil engineering application be decomposed into objects of a given set of classes. Then the set of objects forms the data base of the application. The objects contain attributes and methods. Properties of the objects are stored in the attributes. Algorithms which the objects perform are implemented in the methods of the objects.

If objects are modified by a user, the consistency of data in the base is destroyed. The data base must be modified in an update to restore its consistency. The sequence of the update operations is not arbitrary, but is governed by dependence between the objects. The situation can be described mathematically with graph theory. The available algorithms for the determination of the update sequence are not suitable when the data base is large. A new update algorithm for large data bases has been developed and is presented in this paper.

1 Introduction
A relation on a set of objects in a data base can be visualized as a directed graph. Graph theory can therefore be used for the description of the data base.

The data base of the application often consists of a large number of objects. It is therefore not advisable to update the entire data base after every modification. Delayed updates are used for parts of the data base which are called the update domains.

The aim of our research is to develop efficient algorithms to determine the domain of the update and the sequence of the update operations which restore consistency in a specified part of the data base. The algorithm has to be developed, accounting for dependence of the objects in the data base.

2 Concept of delayed updates
The information of a civil engineering application often changes several times before the final solution is found. It is therefore not advisible to update the objects in the data base of a civil-engineering application after every modification. Modifications are accumulated over a period of time and the objects are then updated with delay.

Three subsets of the data base are considered for updates: the modification set, the goal set and the update domain.

2.1 Level of the modification
In order to determine the modification set, modifications have to be registered. There are three levels for the registration of modifications: attribute level, object level and set level. At a specific level, a modification is registered if an element of that level is modified. Due to the large number of objects of a civil engineering application whose number of attributes may also be large, it is not advisible to consider every single attribute of an object. For civil engineering applications, the attribute level is too “micro” so that the object level has been chosen for the development of the concept. In some instances, it is advisable to use the set level (functional dependence).

2.2 Modification and goal sets

If any attribute of an object is changed, the object is considered to be modified. The modified objects form the modification set. An object can be modified by a user, by its own methods or by the methods of another object. The modified object is registered. Let a data base be considered in time. Let the object set M of a data base be consistent at time t_0. The set of objects whose attributes have been changed since the point in time t_0 is called the modification set of the data base and denoted by $A(t_i)$.

$A(t_i) := \{ x \in M \mid$ An attribute of x has been changed in the period $(t_0, t_i) \}$

The set of objects whose properties are to be updated, accounting for the current modification in the data base, is called the goal set of an update. The goal set can be specified by the user of an application (an engineer, an architect, etc.) or by a method of an object. Consider an object set M in a data base which was consistent at time t_0. Let the modification of the object set be $A(t_i)$. Let a set of objects $Z(t_i)$ be defined whose properties are to be determined at time $t_i > t_0$ accounting for the modification $A(t_i)$. The set $Z(t_i)$ is called the goal set of the update.

$Z(t_i) := \{ x \in M \mid$ The attributes of x are computed in the update at $t_i \}$

2.3 Update domain and sequence of updating

The set of objects whose properties must be updated to restore concistency for given modification and goal sets is called the update domain. Consider an update of an object set M after modification $A(t_i)$ with a goal set $Z(t_i)$. The subset $D \subseteq M$ containing the objects which influence the update is called the domain of the update at point in time t_i and denoted by $D(t_i)$.

The objects in the update domain $D(t_i)$ are updated at time t_i in a specified sequence. An object $x_i \in D(t_i)$ must be updated before an object $x_k \in D(t_i)$ if the object x_k is dependent on the object x_i.

A relation on a set of objects in a data base is visualized as a directed graph. Objects are represented by nodes of the graph. An object x_k is dependent on an object x_i if at least one path exists from the node x_k to the node x_i. An object x_k is dependent on a set X if the object x_k is dependent on at least one object of the set X.

There are different types of dependence graphs which can be constructed to visualize a relation on a set of objects. The type of the graph is determined according to the type of dependence of the objects.

3 Dependence of objects

There are two types of dependence of objects : structural and functional.

3.1 Structural dependence

If the reference of an object B is an attribute of an object A, then the object A is structurally dependent on the object B. The object Triangle t is structurally dependent on the objects Node n1, n2 and n3 (Fig.1).

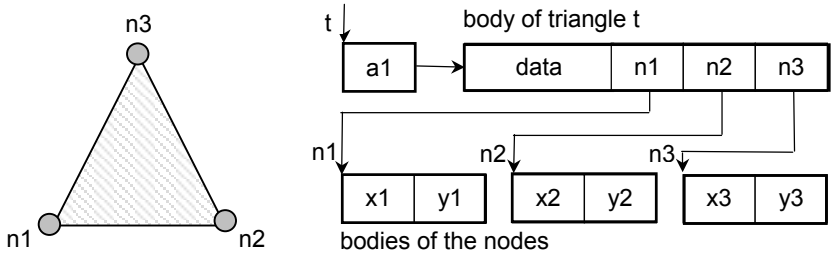

Fig.1 Example of structurally dependent objects

The relation on a set of structurally dependent objects is represented by a directed graph (Fig.2).

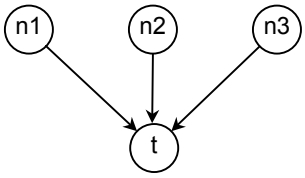

Fig.2 Graph of structurally dependent objects

3.2 Functional dependence

If the object A is an input value and the object B is an output value of an algorithm Z, then the object B is functionally dependent on the object A (Fig.3).

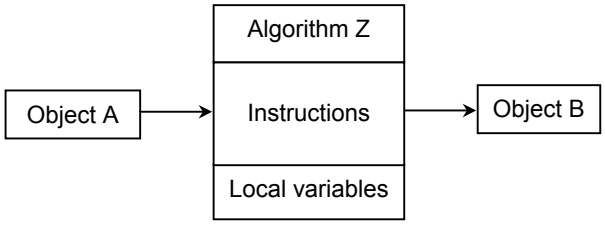

Fig.3 Example of functionally dependent objects

Relations on a set of functionally dependent objects are represented by directed bipartite graphs (Fig.4).

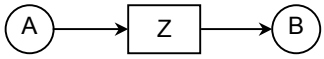

Fig.4 Graph of functionally dependent objects

Structural dependence is static and known at compile-time. Functional dependence is dynamic and determined at run-time. The differences between structural and functional dependence of objects in data bases lead to different update algorithms.

4 Update algorithm for structural dependence

4.1 Subsets used for updates

Three subsets of the data base are considered for an update at point in time t_i : the modification set $A(t_i)$, the goal set $Z(t_i)$ and the update domain $D(t_i)$.

Modifications which have been made by a user (external modifications) over a period of time (t_{i-1}, t_i) are called the increment of the modification set $A(t_i)$ and denoted by $dA(t_i)$:

$dA(t_i) := \{ x \in A(t_i) \mid$ An attribute of x has been changed externally in the period $(t_{i-1}, t_i) \}$

In addition to these three subsets, the subset for accumulation of the updated objects is considered. It contains all objects which have been modified by updating methods (internal modifications) during the time period (t_0, t_i) and are independent of $dA(t_i)$. The internal modification subset is denoted by $R(t_i)$. Internal modifications are accumulated in the subset $R(t_i)$ after every update.

$R(t_i) := \{ x \in (D(t_0) \cup D(t_1) \cup \ldots \cup D(t_{i-1}) \mid x$ is independent of $dA(t_i) \}$

4.2 Reduced subsets

The time and the complexity of updates can be decreased by reducing the subsets used for updates.

$Z_r(t_i)$: reduced goal set containing the objects of $Z(t_i)$ which are dependent on $A(t_i)$.

$D_r(t_i)$: reduced update domain containing the objects of $D(t_i)$ which are not contained in $R(t_i)$.

All objects in $D_r(t_i)$ are updated at t_i. This set is therefore modified internally. After updating, the objects in the sets are added to the set $A(t_i)$ and $R(t_i)$ to form the following extended sets:

$A^*(t_i)$: extended modification set $A(t_i)$ after update.

$R^*(t_i)$: extended set $R(t_i)$ after update.

$A_r(t_i)$: reduced modification set containing the objects of $A^*(t_i)$ that have at least one successor which is not contained in $R^*(t_i)$.

The reduced modification set $A_r(t_i)$ is added to the modification set at the next update at t_{i+1} :

$A_r(t_i) \subseteq A(t_{i+1}) \quad A(t_{i+1}) = dA(t_{i+1}) \cup A_r(t_i)$

4.3 Time sequence of determination of the subsets

The order of determination of the sets and the update at point in time t_i is presented in the following table (Fig.5).

Step	Set	Determination
	t_0 t_{i-1} t_i t_{i+1}	
	$R^*(t_{i-1})$	
	$A^*(t_{i-1})$	
	$A_r(t_{i-1})$	
1	$dA(t_i)$	set by a user
2	$Z(t_i)$	set by a user
3	$R(t_i)$	$= \{x \in R^*(t_{i-1}) \mid x \text{ is independent of } dA(t_{i-1})\}$
4	$A(t_i)$	$= dA(t_i) \cup A_r(t_{i-1})$
5	$Z_r(t_i)$	$= \{x \in Z(t_i) \mid x \text{ is dependent on } A(t_i)\}$
6	$D(t_i)$	$= \{x \in M \mid x \text{ is dependent on } A(t_i) \text{ and } Z_r(t_i)\}$
7	$D_r(t_i)$	$= \{x \in D(t_i) \mid x \notin R(t_i)\}$
8	Updating	
9	$R^*(t_i)$	$= R(t_i) \cup D_r(t_i)$
10	$A^*(t_i)$	$= A(t_i) \cup D_r(t_i)$
11	$A_r(t_i)$	$= \{x \in A^*(t_i) \mid \exists (x,y) : y \notin R^*(t_i)\}$
	$dA(t_{i+1})$	
	$Z(t_{i+1})$	

Fig.5 Time sequence of determination of the subsets

4.4 Example of an update

Let the directed graph represent a relation between objects in a data base. Every node of the graph has three attributes: a name, a number N and a value V. The name of a node is unique. The number is an integer value. It is set by a user. The value is an integer value. It is calculated by an internal method as the sum of the object's number N and the values V_p of the predecessors of the object. Node A is a predecessor of node B if there is a directed arc with start node A and end node B. The symbols which are used in the graphs are explained in Fig.6.

Fig.6 Symbols in the example graphs

Let the graph be consistent at point in time t_0. During every period of time (t_i, t_{i+1}) the user can change the attribute "number" of an object and select an object as a goal for an update. At every point in time t_i the graph must be updated for a selected goal. Consider the object graph in four points of time: $t_0 < t_1 < t_2 < t_3$. The update domain is to be determined. The objects in the update domain are to be updated according to a selected goal at every point in time t_i. The update process is shown graphically in Fig.7 - 10.

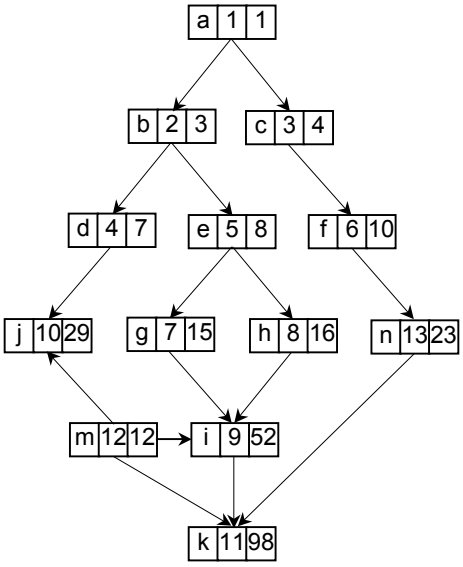

Fig.7 Update at time t_0

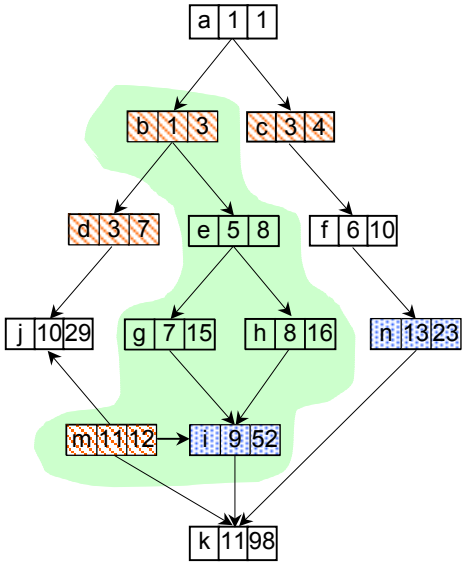

Fig.8 Update at time t_1

a) at point in time t_0

$A(t_0) = \{\varnothing\}$ – the modification set

$Z(t_0) = \{\varnothing\}$ – the goal set

$D(t_0) = \{\varnothing\}$ – the update domain

$R(t_0) = \{\varnothing\}$ – accumulated interior modifications

b) at point in time t_1

$dA(t_1) = \{b, d, m\}$ → $A(t_1) = \{b, d, m\}$

$R(t_1) = \{\varnothing\}$

$Z(t_1) = \{i, n\}$ → $Z_r(t_1) = \{i\}$

$D(t_1) = \{b, e, g, h, m, i\}$ → $D_r(t_1) = D(t_1)$

$R^*(t_1) = \{b, e, g, h, m, i\}$

$A^*(t_1) = \{\mathbf{b}, \mathbf{d}, e, g, h, \mathbf{m}, \mathbf{i}\}$ → $A_r = \{b, d, m, i\}$

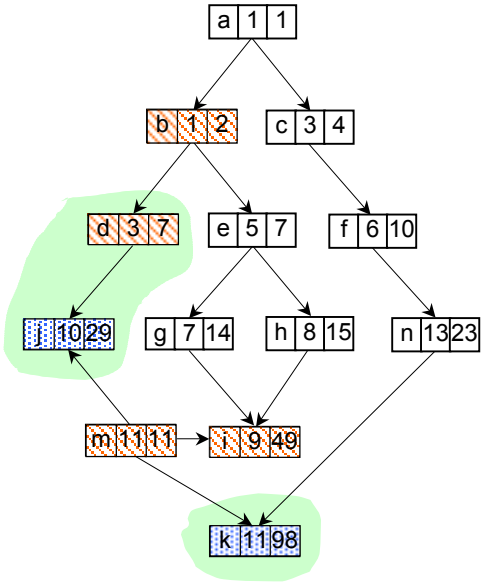

Fig.9 Update at time t_2

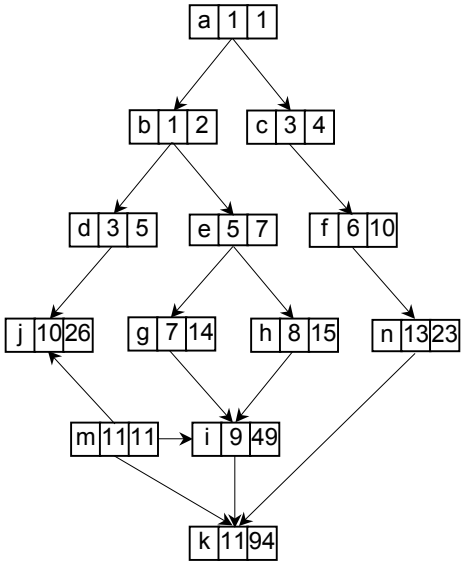

Fig.10 Update at time t_3

c) at point in time t_2

$dA(t_2) = \{\emptyset\}$ → $A(t_2) = \{b, d, m, i\}$
$R(t_2) = \{b, e, g, h, m, i\}$

$Z(t_2) = \{j, k\}$ → $Z_r(t_2) = \{j, k\}$

$D(t_2) = \{b, d, m, j, e, g, h, i, k\}$ → $D_r = \{d, j, k\}$
$R^*(t_2) = \{b, e, g, h, m, i, d, j, k\}$
$A^*(t_2) = \{b, m, i, d, j, k\}$ → $A_r(t_2) = \{\emptyset\}$

d) at point in time t_3

$A(t_3) = \{\emptyset\}$
$R(t_3) = \{b, e, g, h, m, i, d, j, k\}$

$Z(t_3) = \{\emptyset\}$

$D(t_3) = \{\emptyset\}$

4.5 Determination of dependent objects

The task of determination of dependent objects is equivalent to the determination of a path between these objects. The graph algorithm of depth-first search (DFS) is used for the determination of a path between objects (Cormen et al. 2001). Let the resulting set of depth-first search with source set Y be denoted DFS(Y). Then the objects contained in DFS(Y) are dependent on a subset Y of the graph. The intersection of DFS(Y) with a set X contains the objects of a set X which are dependent on a set Y :

$DFS(Y) \cap X = \{x \in X \mid x \text{ is dependent on } Y\}$

4.6 Determination of the update domain and the sequence of updating

The update domain and the sequence of updating can be determined by two DFSes with topological sorting. If the modification set A and the goal set Z are known, then the update domain and the sequence of update are determined as follows:

1. Run the depth-first search from every node of the modification set A.
2. Get the topologically sorted sequence D_1 of DFS.
3. Reduce the goal set Z by intersection with D_1 : $Z_r = Z \cap D_1$
4. Inverse the arcs of the graph.
5. Run the DFS from every node of the reduced goal set Z_r.
6. Get the topologically sorted sequence D_2 of DFS.
7. Get the update domain D as the intersection of D_1 and D_2 : $D = D_1 \cap D_2$.

4.6.1 Example : Update domain and sequence

The algorithm for determination of the update domaine and sequence is explained with the directed graph in Fig.11.

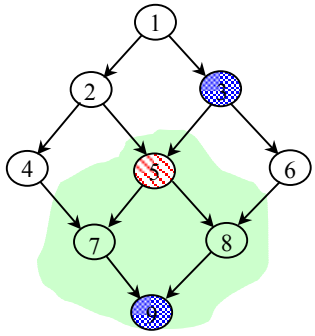

Fig.11 Update domain in the graph

Input data:
A = {5} - modification set
Z = {3, 9} - goal set

1. DFS of the modification set with topological sorting :

5
5 7
5 7 **9** → {9}
5 **7** → {9, 7}
5
5 **8** → {9, 7, 8}
5
set D_1 = {9, 7, 8, 5}

2. Reduction of the goal set by intersection with set D_1 :
$Z_r = Z \cap D_1 = \{9\}$

3. DFS of the reduced goal set in the inverse direction of the arcs with topological sorting :

9
9 7
9 7 5
9 7 5 2
9 7 5 2 **1** → {1}
9 7 5 **2** → {1, 2}
9 7 5
9 7 5 **3** → {1, 2, 3}
9 7 **5** → {1, 2, 3, 5}
9 7
9 7 **4** → {1, 2, 3, 5, 4}
9 **7** → {1, 2, 3, 5, 4, 7}
9
9 8
9 8 **6** → {1, 2, 3, 5, 4, 7, 6}
9 **8** → {1, 2, 3, 5, 4, 7, 6, 8}
9
set D_2 = {1, 2, 3, 5, 4, 7, 6, 8, 9}

Update domain : $D = D_1 \cap D_2 = \{5, 7, 8, 9\}$
Sequence of the update : 5 → 7 → 8 → 9

4.7 Steps of the update algorithm

Let the data base be consistent at point in time t_0. Then $A(t_0)$, $Z(t_0)$, $D(t_0)$ and $R(t_0)$ are empty. At every point in time t_i the update is executed in the following steps :

1. The user modifies objects, starting after the last update at time t_{i-1}. These objects form the increment of modification $dA(t_i)$.
2. The user selects the goal set $Z(t_i)$ for the update.
3. Determine $R(t_i)$ by removing every object from $R^*(t_{i-1})$ which is dependent on $dA(t_i)$:
   $R(t_i) = R^*(t_{i-1}) - (R^*(t_{i-1}) \cap DFS(dA(t_i)))$
4. Determine $A(t_i)$ as union of $dA(t_i)$ and $A_r(t_{i-1})$:
   $A(t_i) = dA(t_i) \cup A_r(t_{i-1})$
5. Reduce the goal set $Z(t_i)$ by removing every object which is not dependent on $A(t_i)$:
   $Z_r(t_i) = Z(t_i) \cap DFS(A(t_i))$
6. Determine the update domain :
   $D(t_i) = DFS^T(Z_r(t_i)) \cap DFS(A(t_i))$
7. Reduce the update domain by removing from $D(t_i)$ every object which is contained in $R(t_i)$:
   $D_r(t_i) = D(t_i) - (D(t_i) \cap R(t_i))$
8. Update every object in $D_r(t_i)$ by recalculating its attributes for the modifications.
9. Extend $R(t_i)$ by adding the updated objects:
   $R^*(t_i) = R(t_i) \cup D_r(t_i)$
10. Extend $A(t_i)$ by adding the updated objects:
    $A^*(t_i) = A(t_i) \cup D_r(t_i)$
11. Reduce the modification set: traverse $A^*(t_i)$ to check every successor of its objects. If all successors of an object are contained in $R^*(t_i)$ then the object must be removed from $A^*(t_i)$.

4.8 Complexity of the algorithm

In order to make the update algorithm efficient, each type of operation which it contains must be efficient. The update algorithm is based on two types of operation : set operations ($\cap$, $\cup$, "difference") and search operations (depth - first search DFS). The algorithms for set and search operations are linear in time (Cormen et al. 2001), (Aho et al. 2001).

The algorithm for depth - first search runs in O(n+a) time, where *n* is a number of the nodes and *a* is a number of the arcs of the graph. The algorithm for set operations runs in O(*n*) time if the sets are represented as sorted lists. The complexity of the update algorithm is therefore :

$max(O(n+a), O(n)) = O(n+a)$.

The graph, the increment of the modification set and the goal set are input data for the update algorithm. Test graphs with *n* nodes and *2n* to *10n* arcs were generated with the following properties (Fig.12) :

- The graph is acyclic.
- There are no separated nodes (nodes without arcs) in the graph.
- The graph is randomly generated as a network with a start node and a target node.

The number of nodes in the modification set and in the goal set is fixed, but the nodes in these sets are selected randomly.

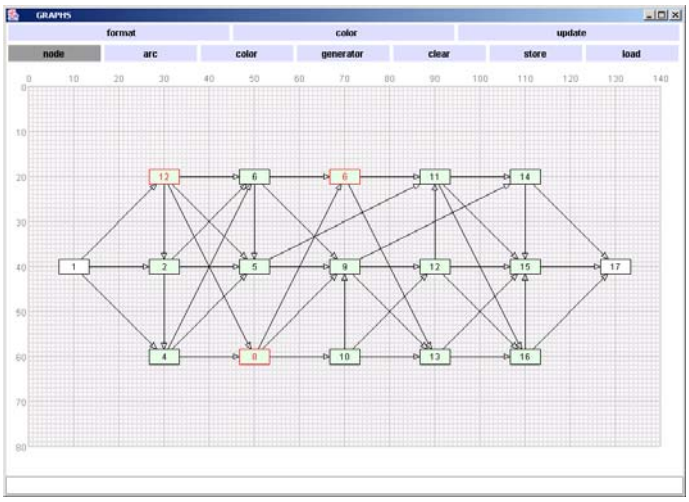

Fig.12 Structure of a test graph

The update domain and the sequence of updating are the output data of the algorithm. The update domain is determined for the input test data: its size is not fixed.

The test has been carried out for a single point in time for steps 1 to 8 of the update algorithm. The number of required simple operations is measured. The simple operation for depth - first search (DFS) is the method *add* in the methods *getAdjacencyListForNode(nodeName)* and *getPredecessorsListForNode(nodeName)*. The simple operation for INTERSECTION is method *compareTo(object)*.

Figures 13 – 17 show the number of operations for the depth - first search in step 3 and the intersection in step 6 of the update algorithm.

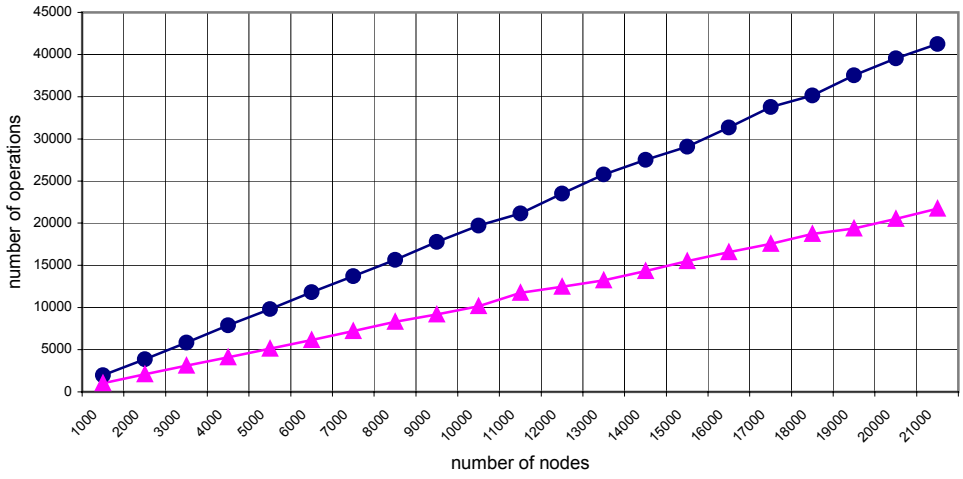

Fig.13 Average number of simple operations
modif. set size = goal set size = 100, number of arcs / number of nodes = 2
depth-first search from dA intersection DFS(Zr) with DFS(A)

The dependence is linear. Every simple operation *add* or *compareTo(object)* runs in constant time O(1). The time depends on properties of the computer. The base operations DFS and INTERSECTION therefore run in linear time (Fig.13).

For different sizes of the modification set and the goal set, the algorithms DFS and INTERSECTION require an equal number of simple operations (Fig.14, 15). This can be explained by the selected sizes of the modification set and the goal set which are much smaller than the number of nodes in the graph.

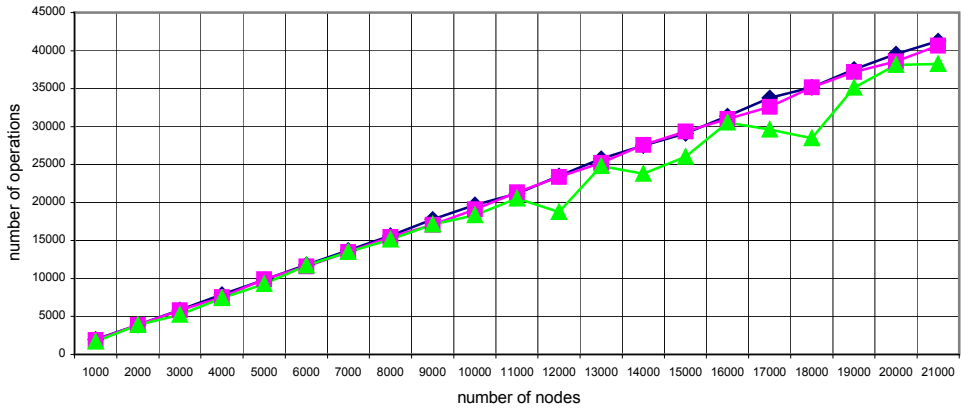

Fig.14 Average number of simple operations for DFS(dA)

modset size = 100 modset size = 50 modset size = 10

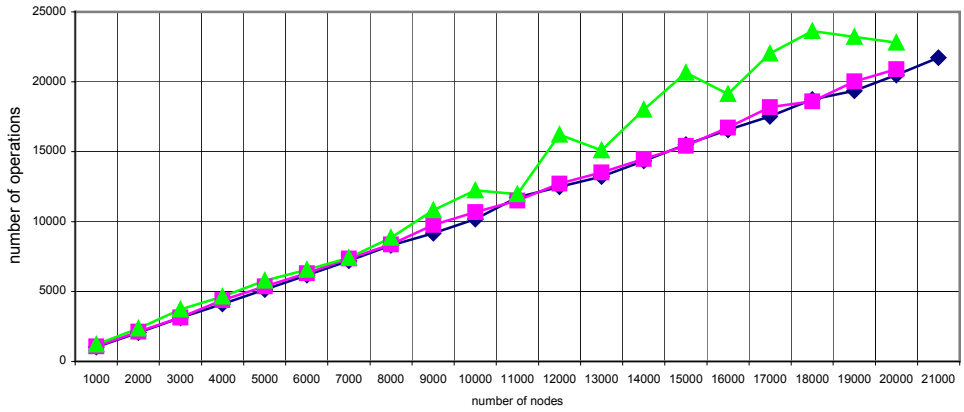

Fig.15 Average number of simple operations for INTERSECT(DFS(Zr),DFS(A))

modset size = 100 goalset size = 100 modset size = 50 goalset size = 50 modset size = 10 goalset size = 10

The number of simple operations in the DFS algorithm depends on the ratio arcs / nodes, because the number of simple operations *add* in the metod *getAdjacencyListForNode (nodeName)* depends on the number of leaving arcs (Fig.16).

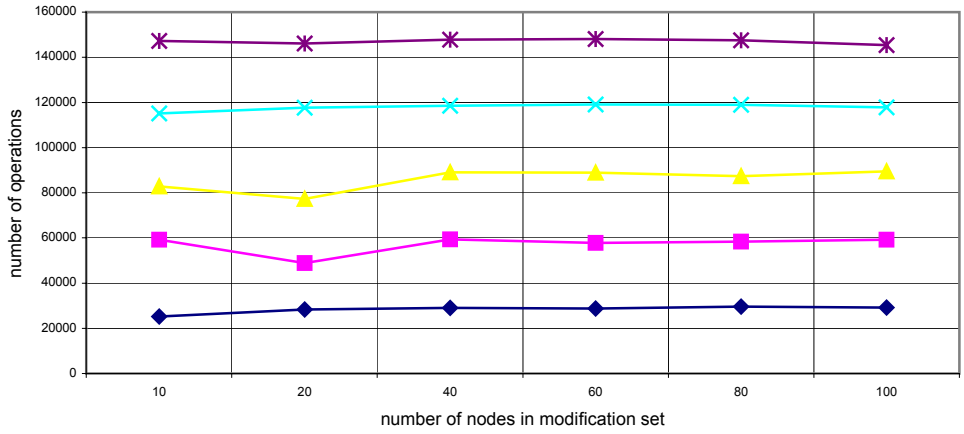

Fig.16 Average number of simple operations for DFS (dA) in a graph with 1500 nodes

a/n = 2 a/n = 4 a/n = 6 a/n = 8 a/n = 10

The number of simple operations in the INTERSECTION algorithm does not depend on the ratio arcs / nodes, because the number of simple operations *compareTo(object)* depends only on the number of nodes in the intersected sets (Fig.17).

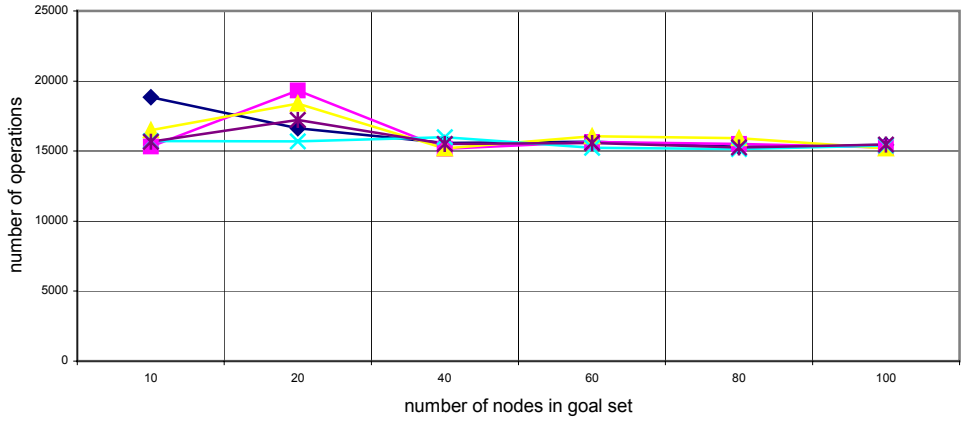

Fig.17 Average number of simple operations for INTERSECTION(DFS(Zr), DFS(A)) in a graph with 1500 nodes

a/n = 2 a/n = 4 a/n = 6 a/n = 8 a/n = 10

The required number of simple operations is independent of the ratio arcs / nodes. This is due to the network structure of the tested graph. The location of the modified nodes and the goal set is selected randomly. Due to the structure of the graph, if a node of the set is located at the begining of the network, then every successive node of the network will be included in the result of the DFS.

5 Update algorithm for functional dependence

The complete dependence graph for functional dependence cannot be set up at the start of the update procedure, since the dependence of objects is only known when the algorithm is executed for specific input values. The algorithm must create the dependence graph as the update procedure progresses. Therefore it is not possible to implement for functional dependence the type of update algorithm which has been presented for structural dependence. The development of update algorithms for functional dependence is in progress.

6 Conclusions

An update algorithm for the case of structural dependence in a data base has been implemented in Java. Its complexity has been tested for large graphs. The algorithm is efficient for the size of data bases which are encounted in engineering praxis. It must be comlemented by the development of an update algorithm for functional dependence, which is more frequently than structural dependence.

7 Acknowledgement

The reported research was conducted in the research project DFG - Gz. PA 162/9-1 "Investigation of Structures in Information Sets of Civil Engineering" of the Deutsche Forschungsgemeinschaft (German Research Foundation DFG). We wish to thank the foundation for its support.

8 References

Pahl, Peter Jan and Beucke, Karl. *Neuere Konzepte des CAD im Bauwesen : Stand und Entwicklung.* Bauhaus-Universität Weimar, Germany : in digital proceedings of „Internationales Kolloquium über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen (IKM)", 2000.

Pahl, Peter Jan. 2002. *Systemtheorie, Grundlagen und Verfahren.* Berlin: Technische Universität Berlin.

Pahl, Peter Jan and Damrath, Rudolph. *Mathematical Foundations of Computational Engineering.* Berlin: Springer Verlag, 2001.

Pahl, Peter Jan. 2001. *Engineering in distributed computer enviroments.* Berlin: Technische Universität Berlin.

Cormen,, Thomas H., Charles E. Leiserson, Ronald L. Rivest and Cliffort Stein. *Introduction to Algorithms.* Second Edition. Massachusetts: MIT Press.,Cambridge, 2001.

Hanff, Jochen. 2003. *Abhängigkeiten zwischen Objecten in ingenieurwissenschaftlichen Anwendungen*. Berlin : Dissertation, Technische Universität Berlin.

Aho, A. V., J. E.Hopcroft and J. D. Ullman. *Data structure and algorithms.* Massachusetts: Addison-Wesley, 1983 (in rus.transcript. 2001).